Comparativo, metodología ágil Crystal y arquitectura dirigida por modelos MDA

John Sebastián Cortés Moreno

Universidad Tecnológica de Pereira
Facultad de Ingenierías
Programa de Ingeniería de Sistemas y Computación
Pereira, Risaralda
2017

Comparativo, metodología ágil Crystal y arquitectura dirigida por modelos MDA

John Sebastián Cortés Moreno

Trabajo de grado presentado como requisito para optar el título de ingeniero en sistemas y computación

Director,
PhD. Guillermo Roberto Solarte Martínez

Universidad Tecnológica de Pereira
Facultad de Ingenierías
Programa de Ingeniería de Sistemas y Computación
Pereira, Risaralda
2017

Agradecimientos

Quiero agradecer a mis padres por su esfuerzo y dedicación por sacarme adelante y cumplir mis sueños, a mi familia por el apoyo incondicional que han tenido conmigo y los valores que me han inculcado para crecer como persona, gracias a mis amigos que me acompañaron y me motivaron en esta etapa de la vida; Julián Melchor, Camilo Diago, Michael, Cristian, Sebastián Cobos y a los demás que no alcanzo a nombrar, por ser excelentes personas y por los buenos momentos y experiencias inolvidables que compartimos, gracias a los profesores por compartir sus conocimientos, quiero dar un agradecimiento muy especial a mi gran amigo Bryan que me acompañó desde el inicio de la carrera y que para mí ha sido un ejemplo de vida, es un honor para mi haber vivido esta etapa junto a él, y por ultimo agradecer a la Universidad porque en ese lugar viví esta grata experiencia, muchas gracias.

John Sebastián Cortés Moreno

Tabla de contenido

Lista de tablas

Lista de figuras

Resumen

En la monografía desarrollada se analizaran detalladamente artículos e información respectiva y pertinente acerca de la arquitectura de software y las metodologías agiles de desarrollo , para principalmente lograr obtener un compilado competente y así lograr acercar al lector a los temas referidos, dicho compilado contara con ítems tales como: el concepto de metodologías agiles, principales metodologías agiles, ventajas y desventajas de las metodologías, también contara con concepto de arquitectura de software, principales arquitecturas de software, ventajas y desventajas de las arquitecturas, técnicas de desarrollo, arquitecturas agiles, se realizara un caso de estudio en donde se podrá observar la implementación de estas dos metodologías en el desarrollo de un proyecto para al final concluir con la importancia de estas prácticas.

Abstract

In this monograph, articles and information regarding the software architecture and the agile development methodologies will be analyzed on detail, mainly to obtain a competent compilation so be able to bring the reader closer to the aforementioned topics, this compilation will have topics such as: the concept of agile methodologies, main agile methodologies, advantages and disadvantages of the methodologies, will also have concepts of software architecture, main software architectures, advantages and disadvantages of architectures, development techniques, agile architectures, a case of study where you can see the implementation of these two methodologies in the development of a project for finally conclude with the importance of these practices.

1. Generalidades

1.1 Título

Comparativo, metodología ágil Crystal y arquitectura dirigida por modelos MDA.

1.2 Introducción

En el presente documento se dará una idea general sobre el proyecto a realizar, abordaremos el problema, buscaremos una buena descripción del mismo para poder determinar varios aspectos sobre el desarrollo de este documento tales como el análisis del problema, los objetivos tanto general como especifico, el alcance y el propósito del mismo.

La principal causa del desarrollo del proyecto es suministrar información y dar tratamiento para hacer una breve comparación entre las metodologías agiles y la arquitectura de software y crear una interrogante acerca de una fusión sobre estos temas. El proyecto se apoyará en documentos y artículos tipos A, B y C [1] que nutran en gran manera el contenido del mismo y se pueda desarrollar de forma coherente y puntual.

1.3 Planteamiento del problema.

En el campo de la ingeniería de software existen dos prácticas, la arquitectura y las metodologías agiles que, aunque son muy usadas dentro del desarrollo de software la documentación acerca de ellas está muy dispersa y descentralizada [2], además la

relación entre ambas no se encuentra lo suficientemente documentada ni formalizada a través de un proceso acorde a la materia.

Se analizaran detalladamente artículos e información respectiva y pertinente acerca de estas dos prácticas, para principalmente lograr obtener un compilado competente y así lograr acercar al lector a los temas referidos, dicho compilado contara con ítems tales como: técnicas de diseño, principales metodologías agiles, ventajas y desventajas de las metodologías, también contara con, definición de arquitectura, el rol de la arquitectura en el desarrollo del software y a manera de comparación observaremos la arquitectura y su rol dentro de las metodologías agiles, más específicamente la metodología Crystal y la MDA (Arquitectura dirigida por modelos). Por último, nos encaminaremos hacia una interrogante, la existencia de arquitecturas agiles.

1.4 Objetivo general

Por objetivo principal tenemos como propósito hacer una comparación entre la arquitectura y metodologías agiles, tomando como ejemplo la arquitectura dirigida por modelos MDA y la metodología Crystal, buscando ventajas y desventajas para poder llegar a concluir cual es más conveniente.

1.5 Objetivos específicos

- Investigar y recolectar información en artículos tipo A, B y C y demás que sean pertinentes para abordar la temática.
- Generar el contenido de estudio producto de la investigación previa.
- Realizar una comparación entre las técnicas investigadas.

- Proponer un caso de estudio para determinar la relación entre la metodología y arquitectura de software.

1.6 Metodología

Para llevar a cabo el desarrollo del primero de los objetivos específicos, es necesario seleccionar una amplia muestra de artículos tipo A, B o C que aporten en gran manera al contenido del proyecto.

Partiendo de los contenidos y conocimientos extraídos de la muestra de artículos seleccionados, se realizará de manera ordenada el cuerpo de estudio y desarrollo de los temas, para de esta manera dar cumplimiento al segundo objetivo específico.

Para realizar el punto tres de los objetivos específicos, se estudiarán de manera más detallada las dos técnicas a comparar, teniendo en cuenta ventajas y desventajas de ambas y tareas o procesos que tengan en común.

Finalmente, para dar cumplimiento al cuarto objetivo específico, se debe tener en cuenta toda la temática desarrollada y la comparación anteriormente presentada para presentar el caso de estudio, que se resumirá con una pregunta general.

2. Marco referencial

2.1 Marco teórico

Desde que empezó el desarrollo de software se consideró como un arte que carecía de métodos o técnicas establecidas para guiarse en la construcción de soluciones de tipo software, estos se diseñaban sin ninguna planeación haciendo que los posibles cambios en los requerimientos generaran un gran impacto, a estos cambios se les llamaba mantenimiento de software y se consideraba una actividad de alto costo, no solo económico también de tiempo, por lo que muchas veces no se realizaba, llevando al fracaso del software, es por esto que en los años 80 gracias al estudio de diversos desarrolladores e investigadores surgen nuevas técnicas como las metodologías de desarrollo agiles y el diseño de la arquitectura de software para contrarrestar este impacto negativo, algunas de estas técnicas, procedimientos, normas y protocolos para desarrollo de software son:

2.1.1 Obtención de requerimientos

Son todas las técnicas que usa el analista o investigador para compilar toda le información necesaria para iniciar el proceso de desarrollo del software, existen diversas técnicas para adquirir los requerimientos, las cuales destacan:

- Entrevistas: Es una conversación formal entre el cliente y el analista, donde el analista extrae toda la información relevante que requerirá el software.
- Cuestionarios: Es una encuesta preferiblemente escrita la cual permite obtener información de un grupo grande de personas, su amplia distribución permite el anonimato, haciendo que se puedan obtener respuestas más honestas [3].

- Tormenta de ideas: Son reuniones con un grupo pequeño de personas las cuales aportan ideas para posteriormente realizar un análisis detallado, se recomienda esta técnica cuando no está claro las necesidades a cubrir [3].

2.1.2 Análisis Costo/beneficio

Es una técnica en la que se comparan los costos monetarios que conlleva la realización de un proyecto de software y los beneficios que se esperan, es un análisis muy importante ya que ayuda a la toma de grandes decisiones como la cancelación o aprobación de un proyecto.

2.1.3 Planificación y control del proyecto de software

Es una técnica en la cual se definen los objetivos del software, el alcance, el tiempo estimado de desarrollo, los recursos y el equipo de trabajo, basándose en los factores anteriores se desglosa el proyecto en diferentes actividades para posteriormente ser asignadas, además se lleva un monitoreo constante para llevar un control en cuanto a presupuestos, tiempo y costos.

2.1.4 Diagramas UML

Por sus siglas en ingles Unified Modeling Language, son diagramas que permiten la representación de un software a través de diferentes puntos de vista, se trata de un grafo en donde los vértices representan elementos del software y los arcos representan la relaciones que existen [4].

Los principales diagramas UML o los más usados son:

Diagrama de clases: Muestra cómo se encuentra estructurado un software y las relaciones que existen entre los objetos de dicho sistema, cada objeto es una clase que contiene nombre, métodos y propiedades.

Diagrama de casos de uso: Muestra como interactúan un conjunto de actores, relaciones y casos de uso de un sistema.

Diagrama de secuencias: Muestra en orden las iteraciones entre instancias de clases, componentes o actores del software.

Diagrama de componentes: Muestra como está dividido un software a través de un conjunto de componentes y el comportamiento del servicio que estos componentes proporcionan y usan a través de interfaces.

Diagrama de despliegue: Muestra la estructura física(hardware) que será utilizada para la implementación del software.

2.2 Marco conceptual

Programa: Secuencia de instrucciones, escritas para realizar una tarea específica en una computadora (Stair, Ralph M., et al. 2003).

Iteración: Técnica de desarrollar y entregar componentes incrementales de funcionalidades de un negocio (Wikipedia, 2017).

.

Software: Conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación (IEEE Computer Society Press, 1993).

Framework: Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar (Wikipedia, 2017).

Desarrollo: Realización sistemática de las actividades de planeación, diseño, codificación, pruebas y lanzamiento de productos de software (Guerrero G, Erazo L, Miranda P, Ante W, 2011).

Caso de uso: Descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso (Wikipedia, 2017).

Requerimientos: Son las características que se desea que posea un sistema o un software (Wikipedia, 2017).

Metodología: Conjunto de procedimientos racionales utilizados para alcanzar el objetivo que rige una investigación científica, una exposición doctrinal o tareas que requieran habilidades, conocimientos o cuidados específicos (Eyssautier de la Mora, Maurice 2006).

Arquitectura: Conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software (Wikipedia, 2017).

Componentes: Elemento de un sistema de software que ofrece un conjunto de servicios, o funcionalidades, a través de interfaces definidas (Wikipedia, 2017).

2.3 Marco metodológico

La metodología se basa en el análisis de las diferentes técnicas definidas, destacando las ventajas y desventajas de cada una según los estudios realizados por otros autores y así resaltar los aspectos más importantes para tener en cuenta en el desarrollo de un software.

2.3.1 Tipo de investigación

Trabajo monográfico el cual recopila información acerca de distintos trabajos investigativos ya realizados, para a partir de ellos generar una propuesta de arquitectura ágil o aclarar de modo simple el desarrollo de software aplicando metodologías agiles y arquitectura de software. El tipo de investigación y recopilación de información es cualitativa, exploratoria y descriptiva. Debido a que se busca, se analiza y se describe cada una de las técnicas.

2.3.2 Técnicas para la recolección de la información

En esta monografía se recopiló información de distintas fuentes como Dialnet, IEEE Xplore Digital Library y el repositorio de la UTP, que tuvieran relación con desarrollo de software, metodologías agiles y arquitectura de software, principalmente tesis de grado y artículos de revistas investigativas ya que estas poseen una mejor estructura, entendimiento y desarrollo.

2.3.3 Hipótesis

Será posible articular el diseño de la arquitectura de software con la implementación de alguna metodología ágil, es decir que la arquitectura se elabore evolutivamente conforme avance el desarrollo del proyecto de software para mejorar la calidad de éste.

2.3.4 Población

El análisis, estudio y recopilación de la información, va dirigido hacia las personas que se encuentran en el entorno del desarrollo de software. A través del estudio se interesa conocer los efectos de estas técnicas para así incurrir en posibles beneficios sobre el producto final que en este caso sería el software y garantizar la satisfacción del cliente.

3. Estado del arte

3.1 Antecedentes

3.1.1 En los años 60: No existían metodologías de desarrollo, los sistemas se desarrollaban en corto tiempo, los clientes quedaban insatisfechos y los sistemas carecían de documentación.

3.1.2 En los años 70: Surge el desarrollo en cascada, en el cual un sistema se desarrolla en fases secuenciales, era muy difícil realizar cambios al software, las fases de desarrollo requerían bastante tiempo y se presenta un exceso de documentación.

3.1.3 En los años 80: Surgen metodologías para contrarrestar las falencias del desarrollo en cascada, estas metodologías promueven la planeación antes de desarrollar, la participación del cliente durante el desarrollo y la implementación de prototipos.

3.1.4 En los años 90: Surge el desarrollo ágil el cual se enfoca en mejorar las metodologías anteriores, el desarrollo se hace evolutivo con iteraciones en corto plazo, se reduce la documentación y se valora la opinión de los clientes [8].

3.2 Metodologías agiles

Las metodologías agiles son un conjunto de técnicas y estrategias para desarrollar software de forma flexible evolutiva, adaptativa y orientadas a las personas, se apoyan en los principios del manifiesto ágil, un documento realizado por diversos autores y promotores del desarrollo ágil:

“Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas.

Software funcionando sobre documentación extensiva.

Colaboración con el cliente sobre negociación contractual.

Respuesta ante el cambio sobre seguir un plan.

Esto es, aunque valoramos los elementos de la derecha valoramos más los de la izquierda" [7].

De lo anterior sale como resultado también doce principios:

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia [7].

De lo anterior, se puede afirmar que las metodologías de desarrollo ágil surgen como una reacción a los métodos de desarrollo tradicionales [5] puesto que estos son considerados rígidos sin capacidad de realizar cambios ni actualizaciones, con extensa documentación y poca comunicación tanto entre los miembros del grupo de desarrollo como con los usuarios finales(clientes).

Las metodologías agiles se caracterizan por dividir un proyecto de software en subproyectos más pequeños tomando diferentes características del proyecto original, entregando cada subproyecto en periodos pequeños de entre 4 a 6 semanas construyendo acumulativamente el proyecto original, los equipos de desarrollo son pequeños entre 5 y 10 personas, además se promueve la comunicación entre los miembros del equipo, el cliente es considerado parte del equipo de desarrollo ya que puede aportar ideas o nuevos cambios en los requerimientos, la retroalimentación por parte del cliente es muy importante ya que mejora el proceso y el producto [5].

En la siguiente tabla se puede observar las principales características que debe tener una metodología de desarrollo para considerarse ágil:

Entorno del proyecto		Características del proyecto	
Categoría	Variable	Ágil	No ágil
Equipo de trabajo	Estilo de comunicación	Colaboración regular	Sólo lo necesario
	Localización	Centralizado	Distribuido
	Tamaño	<50 personas	>50 personas
	Aprendizaje continuo	Activo	Olvidado
Gestión del proyecto	Cultura de gestión	Dinámica	Ejecución y control
	Participación	Obligatoria	Rechazada
	Planificación	Continua	Al inicio
	Retroalimentación	Varios mecanismos	Inexistente
El cliente	Implicación	A través del proyecto	Durante fase de análisis
	Disponibilidad	Fácilmente accesible	Difícil
Procesos y herramientas	Input del equipo	Tienen la última palabra	Usan lo indicado
	Cantidad	Las necesarias	Mas de lo necesario
	Adaptabilidad	Puede cambiar	Tal vez no cambie
El contrato	Fechas y requisitos	Flexible	Rígido
	Coste	Según recursos	Fijo

Tabla 1. Comparación entre metodologías agiles y no agiles. Tomada de: https://e-archivo.uc3m.es/bitstream/handle/10016/10757/PFC-ARQUITECTURA_AGIL_DESARROLLO.pdf#chapter.5

En la siguiente tabla se puede observar una comparación entre las metodologías agiles y las metodologías tradicionales de desarrollo:

Metodologías agiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 2. Diferencias entre metodologías agiles y metodologías tradicionales. Tomada de [6].

La aparición de las aplicaciones web y móviles y su gran acogida ha hecho que las empresas desarrolladoras implementen cada vez mas este tipo de metodologías a tal punto de no solo usar una, sino sacar características de varias y mezclarlas [9] ya que aplicarlas trae beneficios como mejorar la calidad del producto puesto que en cada

entrega se hacen revisiones acerca del funcionamiento, aumenta la satisfacción del cliente gracias a su participación, se reducen los riegos y el impacto de los cambios, el proceso de desarrollo se hace más rápido y optimo, algunas metodologías agiles son Scrum, XP, DSDM y FDD.

3.2.1 Metodología Scrum

Es una metodología de desarrollo para productos complejos la cual se caracteriza por entregas parciales, iteraciones, revisiones continuas y trabajo en equipo, se basa en tres pilares empíricos que son transparencia es decir que todos los responsables pueden observar los procesos de desarrollo, inspección quiere decir que hay un constante monitoreo en el desarrollo para evitar variaciones indeseables y adaptación permitiendo que el desarrollo puede aceptar cualquier tipo de cambios [10].

Figura 1. Metodología Scrum. Tomada de: [3]

En el equipo de trabajo de Scrum se cuenta con tres roles que son:

Product owner: Es el dueño del producto, el representante de los interesados del proyecto, se encarga de definir los requisitos del proyecto, definir los tiempos de entrega y revisar las entregas del desarrollo.

Desarrolladores: Es el equipo de trabajo y se encargan del proceso de diseñar o producir el proyecto.

Scrum máster: Es el guía del equipo de desarrolladores, además es el que facilita la comunicación entre el product owner y los desarrolladores [10].

En el desarrollo se presentan 3 eventos los cuales son:

Sprint: El sprint es cuando se entrega parte del proyecto con ciertas características definidas en el sprint backlog, cada sprint empieza inmediatamente termina otro.

Planeación del sprint: Es cuando el equipo de trabajo se reúne a definir los objetivos, la duración y el alcance del próximo sprint.

Daily Scrum: Es una reunión diaria del equipo de trabajo en donde el equipo de trabajo revisa lo que se ha hecho en el día.

Sprint review: Es una reunión que se hace al final de cada sprint en la cual el scrum máster y el product owner revisan lo que se hizo en el sprint y dado el caso hacer cambios o aportar ideas para el siguiente sprint.

Sprint retrospective: Es una reunión que se hace entre el sprint review y la planeación de cada sprint, en esta se revisa que se hizo bien y como mejorar las dificultades que se presentaron para el siguiente sprint [10].

En scrum se manejan los artefactos que son documentos los cuales todos los miembros del equipo deben conocer durante el desarrollo, esto para garantizar los tres pilares de scrum, estos son:

Product backlog: Es la lista de requerimientos y tareas que se necesitan para el producto final.

Sprint backlog: Es la lista de requerimientos y tareas que se necesitan para cada sprint.

Increment: Es el acumulado de todos los requerimientos completados del product backlog [10].

Como ventajas se tiene que gracias a sus reuniones diarias se pueden identificar y corregir fácil y rápidamente los errores en el desarrollo ayudando a la organización a ahorrar tiempos y costos, además en estas reuniones se pueden solucionar problemas entre los miembros de trabajo y por último se tiene que cualquier cambio en los requerimientos puede adaptarse fácilmente, en cuanto a sus desventajas se presentan dificultades al momento de agregar nuevas funcionalidades si no se ha definido una fecha de entrega, además hay un impacto negativo en caso de que un miembro del equipo de trabajo abandone el proyecto [12].

3.2.2 Metodología XP

Extreme programming o XP es una metodología ágil que consiste en diseñar y programar los más rápido posible, esto hace que en muchas veces se salte la documentación [30], se enfoca en proyectos pequeños o medianos, se identifica por

exaltar el trabajo en equipo y el bienestar del equipo de trabajo, así como también la buena comunicación entre todos los miembros interesados en el desarrollo y preocuparse por que el cliente quede satisfecho, además de ser la indicada para proyectos con requisitos cambiantes [11].

Figura 2. Metodología XP. Tomada de: https://es.linkedin.com/pulse/metodolog%C3%ADas-agile-cu%C3%A1l-es-la-mejor-rogelio-toledo-garc%C3%ADa

XP cuenta con diferentes roles dentro del equipo de trabajo, cada uno con responsabilidades diferentes para el desarrollo del proyecto, estos son:

Programador: Es el encargado de desarrollar el proyecto, este debe estar en constante comunicación con el cliente para que el cliente siempre esté aportándole ideas y así quedar satisfecho con el desarrollo.

Cliente: Es el que conoce los objetivos del proyecto, este debe estar en constante comunicación con el programador para guiarlo en su desarrollo.

Tester: Es el encargado de realizar las pruebas al proyecto.

Tracker: Es el encargado de hacer seguimiento al proyecto, verificando que todo se esté haciendo bien y en el tiempo estimado [13].

En esta metodología se presentan las siguientes fases:

Exploración: Es la fase inicial del desarrollo y en esta se redactan las historias de usuario y se define el alcance del proyecto.

Planeamiento del lanzamiento: En esta fase se da prioridad a cada historia de usuario y se establecen la duración que tendrá cada entregable.

Iteración: En esta fase se desarrollan las historias de usuario, aquí la participación del cliente es fundamental ya que se desarrolla el proyecto de acuerdo con sus expectativas.

Testeo: Se hacen las pruebas pertinentes para encontrar errores o inconformidades con el proyecto.

Lanzamiento: Se hacen entrega del software totalmente funcional al cliente.

Como ventajas se tiene que la tasa de errores es baja ya que su programación es muy organizada, se promueve la comunicación entre el equipo de desarrollo y el cliente, además el cliente tiene control sobre las prioridades del proyecto [12], su principal desventaja es que en ocasiones la documentación no se realiza o es muy pobre lo que dificulta el entendimiento por parte de otros interesados en el proyecto.

3.2.3 Metodología DSDM

Dynamic system development methods en español método de desarrollo de sistemas dinámicos es una metodología ágil que se utiliza para la realización de proyectos de software que requieran un tiempo corto de desarrollo, por lo general se usa para proyectos pequeños, su principal objetivo es definir primero tiempo y los costos, una vez fijados, se definen las funcionalidades que se pueden implementar en el producto [8].
En esta metodología destacan los siguientes roles:

Programadores: Encargados del desarrollo del proyecto.

Coordinador Técnico: Encargado de que se cumplan todas las normas técnicas al momento de desarrollar el proyecto.

Usuario embajador: Es el representante de los interesados en el proyecto, se encuentra presente en el desarrollo del proyecto y se encarga de difundir a los interesados el progreso de este [8].

Visionario: Se encarga de revisar que el proyecto no se salga de contexto con los requerimientos establecidos [8].

Patrocinador del proyecto: Es el responsable del presupuesto del proyecto.

Como ventajas se tiene que esta metodología asegura desarrollos rápidos, la participación del cliente es constante lo que garantiza su satisfacción y la calidad del proyecto, como desventaja se tiene que es una metodología poco común lo que dificulta su implementación, usabilidad y entendimiento [14].

3.2.4 Metodología FDD

Feature Driven Development en español desarrollo basado en funciones es una metodología ágil que se enfoca en dividir un proyecto de software en pequeñas entregas las cuales se monitorean constantemente por parte del cliente y el líder del proyecto [18], además estas pequeñas entregas son totalmente funcionales y se entregan en periodos de tiempo bastante cortos.

El proceso de desarrollo en esta metodología se divide en cinco fases las cuales son:

Desarrollar el modelo general: Es la fase inicial y el equipo de trabajo se divide en pequeños equipos los cuales proponen un modelo basado en los requerimientos y el objetivo del proyecto, una vez terminadas todas las propuestas se escoge la mejor y se inicia con el desarrollo.

Construir una lista de características: En esta fase el equipo de desarrollo construye una lista de características o funcionalidades las cuales deben ser evaluadas y entendidas por el cliente [18].

Planear por características: Una vez lista la lista de características se organizan prioritariamente y se asignan al programador jefe [18].

Diseñar por características: En esta fase se definen que características irán en cada entrega.

Construir: Se procede a desarrollar el proyecto por entregas hasta tenerlo totalmente completo.

En FDD se presentan diversos roles que son:

Project manager: Es el encargado de reportar el progreso y administrar los presupuestos del proyecto.

Arquitecto: Es el responsable del diseño general del proyecto.

Gerente de desarrollo: Monitorea las actividades de desarrollo.

Jefe de programadores: Asigna tareas y orienta a los demás programadores.

Expertos del dominio: Representante del cliente el cual conoce los objetivos y requerimientos del proyecto.

Como ventajas se tiene que en esta metodología se presentan pequeñas entregas funcionales las cuales se desarrollan colaborativamente entre los clientes y los desarrolladores, estas entregas se monitorean constantemente lo que tiene como resultado entregas de alta calidad, como desventajas se tienen que los plazos de entregas son muy cortos y no existe buena documentación a cerca de esta metodología.

3.3 Arquitectura de software.

La arquitectura de software es una perspectiva dividida, cada una con diversas características que componen un sistema, la cual incluye todos sus componentes, como funcionan y cómo interactúan entre sí para alcanzar con el propósito del sistema [15], se entiende los componentes como objetos, clases, nodos etc. Que surgen de la abstracción y son pertenecientes al software, se crea en una etapa previa al desarrollo y sirve de guía para su diseño [16], además muestra todos los requisitos funcionales y como se cumplen,

hoy en día los proyectos de software tienden a ser grandes haciendo que los grupos de trabajo estén compuestos por diferentes personas cada una con intereses distintos, es por eso que es muy importante que todos los miembros del equipo de trabajo conozcan la arquitectura de software para conocer cuál es el alcance de sus actividades y responsabilidades dentro del proyecto.

Arquitecto de software: El arquitecto tiene un papel fundamental dentro de la arquitectura de software pues es el encargado de diseñar el proyecto desde una manera abstracta, además también se encarga de escoger las herramientas y formar el equipo de trabajo para el desarrollo, debe tener ciertas habilidades y aptitudes de las cuales destacan:

- Buenas bases en desarrollo de software.
- Ser líder y tener buen dominio de la comunicación.
- Ser ordenado.
- Experiencia en negocios.
- Buen manejo de presupuestos.

El arquitecto debe realizar ciertas actividades dentro del proceso de desarrollo del proyecto, las cuales destacan:

- **Generación del proyecto:** En esta actividad el arquitecto traduce una serie de necesidades propuestas por un cliente en un proyecto de software.
- **Evaluar requerimientos:** En esta actividad el arquitecto debe enfocarse en que todos los requerimientos se cumplan de acuerdo con el proyecto propuesto.

- **Diseñar el sistema:** En esta actividad el arquitecto debe poner en practica todos sus conocimientos para poder estructurar de forma abstracta el proyecto, además debe hacer que los demás interesados del proyecto la entiendan.
- **Guiar el desarrollo del proyecto:** En esta actividad el arquitecto debe orientar a los desarrolladores del proyecto, resolviendo dudas y asegurándose de que se cumpla con todos los requerimientos y atributos de calidad establecidos [27].

Patrones arquitectónicos: Son estructuras reutilizables, predefinidas que se pueden utilizar en determinados contextos, estos surgen de la experiencia de otros desarrolladores al enfrentarse a problemas de software comunes.

Vistas arquitectónicas: Son un conjunto de descripciones simplificadas o abstracciones de un sistema cada una diseñada desde una perspectiva diferente.

Las fases de la implementación de la arquitectura de software dependen de la metodología de desarrollo que se use, las presentes en la mayoría de metodologías son [16]:

Levantar requerimientos: En esta fase se capturan los requerimientos por parte del usuario final, se analizan y se documentan.

Diseñar: En esta fase se define las estructuras que componen la arquitectura, basándose en patrones de diseño, tácticas de diseño y elecciones tecnológicas, esta es la fase más compleja del proceso.

Publicar: Es la fase donde se da a conocer el diseño al equipo de trabajo y a los interesados del proyecto, la arquitectura debe estar bien documentada para mejorar el entendimiento [16].

Evaluar: En esta fase se analiza el diseño y la documentación para corregir posibles errores antes de proceder con el desarrollo, esto se hace para evitar perdidas de costo y tiempo.

3.3.1 Arquitectura de software orientada a servicios

La arquitectura orientada a servicios (SOA) permite que una aplicación pueda descomponer su propósito principal en un conjunto de servicios para que sean consumidos por los usuarios [17] permitiendo la reutilización de estos servicios, dicho esto podemos decir que facilita el acceso de usuarios a las funcionalidades de los diversos componentes que integran un software, en la siguiente imagen se puede observar un comparativo de antes y después de implementar SOA:

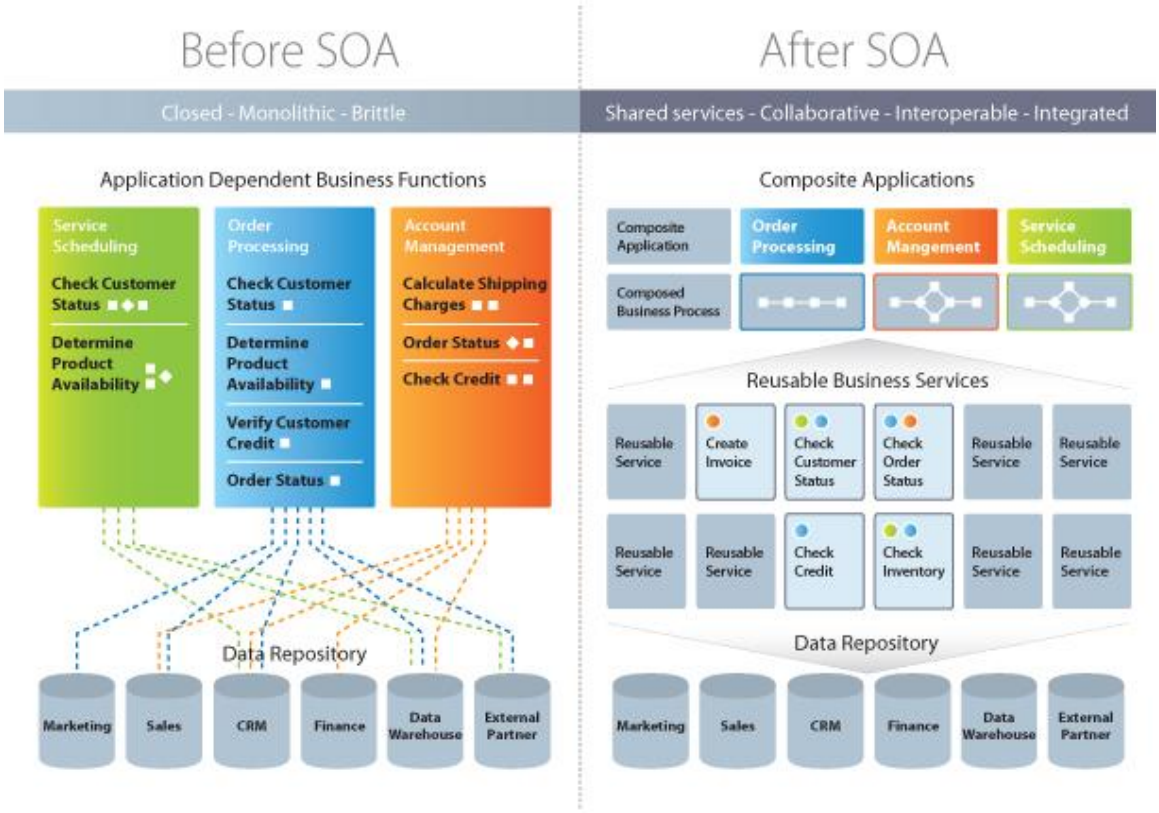

Figura 2. Arquitectura de software. Tomada de: http://www.tridens.si/expertise/soa/

Características principales:

- Servicios reutilizables.
- Puede empaquetar procesos de negocio como servicios [17].
- Los clientes y otros servicios pueden acceder a otros servicios locales ejecutándose en el mismo nivel [17].
- Los clientes pueden acceder a servicios de diferentes procesos de negocio.
- Datos precisos y oportunos.

Como ventajas se tienen que al poseer servicios reutilizables la organización puede obtener una reducción de costos, los servicios son autónomos y se puede acceder a ellos de forma fácil es independiente [17], por desventaja se tiene que requiere de una gran inversión para la implementación además no se recomienda para aplicaciones de corto tiempo de vida.

3.3.2 Arquitectura cliente-servidor

La arquitectura cliente servidor permite que diversos clientes (equipos remotos) envíen y revivan solicitudes de servicios a un servidor (equipo host), generalmente esta arquitectura presenta a los clientes una interfaz agradable y de fácil entendimiento con un nivel de transparencia alto, esto quiere decir que el cliente no conoce las características del servidor como lo son software, hardware y su ubicación, además de que todas las operaciones se realizan del lado del servidor, en la siguiente imagen se puede observar la estructura de esta arquitectura:

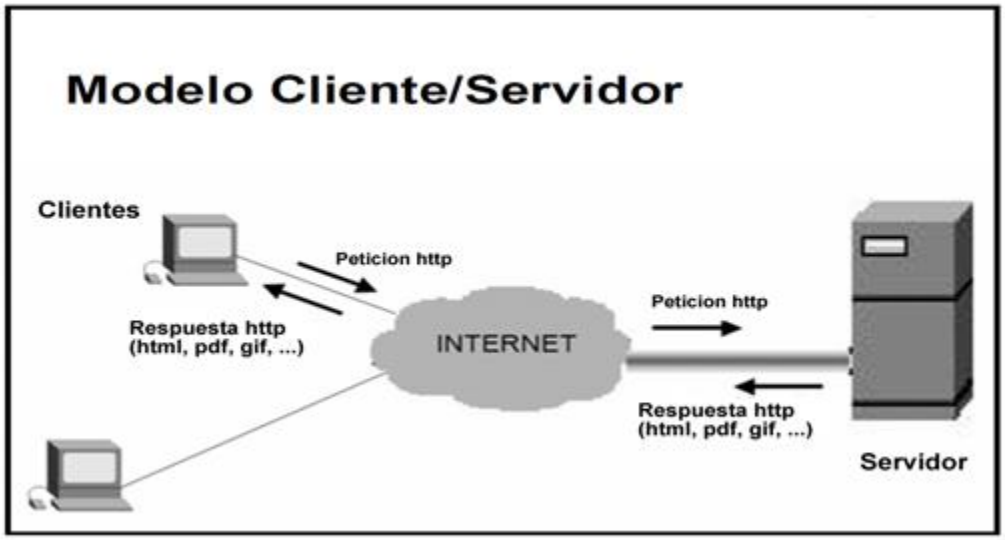

Figura 2. Arquitectura de software. Tomada de:
http://tema3losblogs.blogspot.com.co/2014/12/arquitectura-cliente-servidor.html

Características principales:

- Se compone de cliente, un servidor, un middleware que es el intermediario entre el cliente y el servidor y una red que los conecta.
- Usa diferentes protocolos de comunicación de información [17].
- El cliente envía una petición, el servidor la procesa y le envía la respuesta al cliente.

Como ventajas se tiene que es una arquitectura escalable por lo que si se agregan más clientes o servidores no afectan al resto de la arquitectura, la mayoría de datos se almacenan en servidores los cuales deben manejar altos estándares de seguridad, además los procesos se realizan en los servidores que son maquinas potentes lo que hace que esta arquitectura sea muy eficaz, como desventajas se tienen que el montaje y mantenimiento de la arquitectura es bastante costoso, la congestión de peticiones

puede afectar al servidor y una posible falla en algún servidor puede afectar toda la arquitectura.

3.3.3 Arquitectura limpia

La arquitectura limpia consiste en dividir un proyecto de software generalmente en 4 capas que son: entidades, casos de uso, adaptadores de interfaz y frameworks y dirvers, además se propone una regla de dependencia entre ellas que va de afuera hacia dentro [19] como se puede ver en la siguiente imagen:

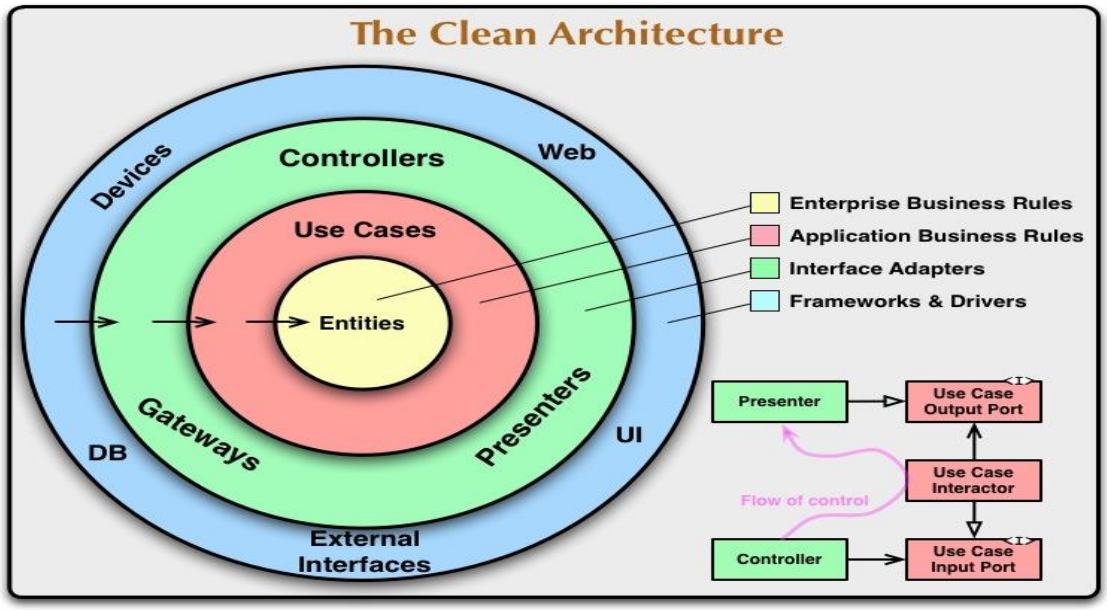

Figura 3. Arquitectura limpia. Tomada de [21].

Entidades: En esta capa se encuentran las reglas del negocio de la empresa [21], son los datos mas importantes del proyecto y son los que probablemente nunca se vean afectados por cambios.

Casos de uso: En esta capa se encuentran las reglas específicas de la aplicación [21], en esta capa se extraen los datos de las entidades para cumplir y darle forma a los casos de uso.

Adaptadores de interfaz: Esta capa se encarga de convertir los datos de las capas anteriores en datos entendibles para la capa de frameworks y drivers.

Frameworks y drivers: Es la capa mas externa y en esta van todos los frameworks, drivers y vistas, además en esta capa se presentan los datos de manera legible al usuario final.

Características principales:

- Independiente de frameworks.
- Testeable.
- Independiente de interfaces de usuario UI.
- Independiente de bases de datos [21].

Como ventajas se tiene que es una arquitectura que tiene una gran tolerancia a los cambios de requisitos y tecnologías, también presenta facilidad de implementación, testeo y mantenimiento, por desventaja se tiene que es una arquitectura relativamente nueva por lo que hay poca documentación acerca de esta.

4. Metodología ágil para el estudio.

4.1 Metodología crystal

La metodología crystal es un conjunto de metodologías de desarrollo de software las cuales están clasificadas según el tamaño y la complejidad del proyecto, entre mas oscuro el color del nombre mayor número de personas implicadas en el desarrollo, un mayor tamaño del proyecto y, por lo tanto, la necesidad de mayor control en el proceso [22], esta metodología hace prioridad a la buena comunicación entre los miembros del equipo de trabajo y en reducir la documentación.

		Crystal Methodologies				
		Clear	Yellow	Orange	Red	Maroon
Criticality of the Project	Life (L)	L6	L20	L40	L80	L200
	Essential Money (E)	E6	E20	E40	E80	E200
	Discretionary Money (D)	D6	D20	D40	D80	D200
	Comfort (C)	C6	C20	C40	C80	C200
		1 to 6	7 to 20	21 to 40	41 to 80	81 to 200
		Number of People involved in the Project				

Figura 4. Clasificación de metodologías crystal. Tomada de: https://www.linkedin.com/pulse/crystal-self-awareness-balaji-sathram-pmi-acp-csp-csm-cspo-

En la imagen anterior podemos observar los elementos que componen la criticidad o complejidad de un proyecto en crystal en caso de que este falle, estos elementos son:

L: Perdida de alguna vida.

E: Perdida de dinero significativa, puede afectar la continuidad del proyecto [23].

D: Perdida de dinero no significativa.

Confort: Perdida de confort.

Además, podemos ver las metodologías que componen el conjunto de crystal:

Clear: Implementada para proyectos muy pequeños de entre 1 a 6 personas y con nivel de criticidad bajo, esta es la más usada porque ofrece un mayor control del proyecto.

Yellow: Implementada para proyectos pequeños de entre 6 a 20 personas y con nivel de criticidad no tan bajo.

Orange: Implementada para proyectos medianos de entre 21 a 40 personas y con nivel de criticidad medio.

Red: Implementada para proyectos grandes de entre 41 a 80 personas y con nivel de criticidad alto.

Maroon: Implementada para proyectos muy grandes de entre 81 a 200 personas y con nivel de criticidad muy alto, esta metodología es de las menos usadas en este conjunto ya que son pocos los proyectos de esta magnitud.

Propiedades de crystal:

Existen diversas propiedades que aplican para todas las metodologías de crystal, estas son:

- **Entregas frecuentes:** Entregas funcionales en periodos de tiempo corto.
- **Mejora reflexiva:** Reuniones del equipo de trabajo en donde se comunica que se hizo bien y que hay por corregir.
- **Comunicación cercana:** Comunicación rápida y oportuna en el equipo de trabajo.
- **Seguridad personal:** Fomentar la buena comunicación entre los miembros del desarrollo.
- **Enfoque:** Los miembros del desarrollo deben tener las prioridades claras dentro del proyecto.
- **Fácil acceso a usuarios expertos:** Fácil comunicación con los usuarios expertos en caso de requerirlos.
- **Pruebas de automatización, gestión de configuración e integración frecuente:** Facilidad de testear por diferentes usuarios [23].

Principales características:

- Retroalimentación y gran participación por parte del usuario final.
- La arquitectura y los diseños se realizan conforme avanza el proyecto.
- Entregas en periodos de tiempo cortos.
- Prioriza a las personas por encima de los procesos y la documentación [24].
- Fácil adaptación a los cambios.

En crystal como en las otras metodologías se manejan roles los cuales son:

Sponsor: Es el encargado de proporcionar el presupuesto del proyecto.

Experto de negocio: Identifica los objetivos del proyecto.

Usuario experto: Proporciona los casos de uso y los actores del proyecto.

Desarrollador principal: Es el líder del proceso de desarrollo, guía a diseñadores.

Diseñador-Programador: Encargados de desarrollar(codificar) el proyecto.

Como ventaja se tiene que la gran participación del cliente final en el proceso de desarrollo asegura que el proyecto cumple con todos los requisitos y necesidades establecidas, los desarrollos son rápidos y se adaptan fácil a los cambios, por desventaja se tiene en muchas ocasiones obliga a que los miembros del equipo de trabajo trabajen juntos en el mismo lugar, lo que muchas veces se dificulta.

5. Arquitectura para el estudio.

5.1 Arquitectura dirigida por modelos MDA.

La arquitectura dirigida por modelos fue propuesta y patrocinada por el object manager group y tiene como objetivo diseñar una solución a un problema de software expresando sus especificaciones o requerimientos como modelos que son una descripción de un sistema de forma visual, se debe mencionar que en esta arquitectura los modelos no son artefactos del proceso de desarrollo, el modelado es conocido como una de las estrategias básicas del desarrollador para entender un problema y proponer una solución [25], busca garantizar la portabilidad, interoperabilidad y reusabilidad.

En el proceso de desarrollo de esta arquitectura se manejan tres modelos para generar el código y posteriormente darle solución al proyecto, estos son CIM PIM y PSM.

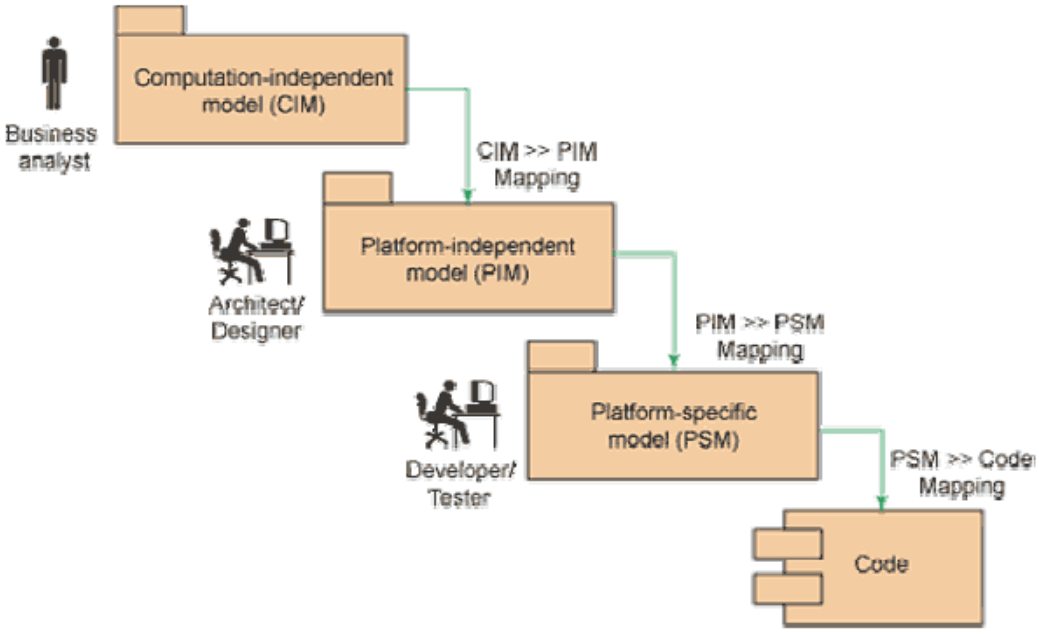

Figura 5. Modelos de la arquitectura MDA. Tomada de:
http://www.jot.fm/issues/issue_2006_03/column4/

Modelos MDA:

CIM: Computation independent model en español modelo independiente de la computación este modelo ayuda a entender el contexto y el dominio del negocio en que va a estar el software, en este modelo se expresa lo que se va a hacer independientemente de como se va a hacer [25].

PIM: Platform-Independents model en español modelo independiente de la plataforma en este modelo se describe como va a ser la solución de software, pero no se define en que plataforma se desarrollara.

PSM: Platform-Specific model en español modelo especifico de plataforma, se basa en el anterior, pero en este si se define en que plataforma de hará el desarrollo.

Fases del desarrollo de un proyecto con MDA:

1. Construir el CIM basado en los objetivos, el negocio y el entorno en el cual estará el proyecto.
2. Basándose en el CIM se debe construir el PIM para describir el sistema.
3. Identificar la plataforma y herramientas que se usaran para el desarrollo.
4. Basándose en el PIM, la plataforma y las herramientas construir los modelos PSE.
5. Basándose en los PSE se procede a codificar y desarrollar el proyecto [25].

Principales características:

- Independencia de plataformas.
- Diferentes modelos, permitiendo ver el software desde diferentes puntos de vista.
- Reusabilidad de modelos reduciendo costos y tiempo.
- Buena abstracción de componentes.

Como ventaja se tiene que gracias a su diseño por modelos facilita el desarrollo, mantenimiento y el entendimiento del software además muchos de estos modelos son reutilizables para otros proyectos lo que reduce costos y tiempo, por desventaja se tiene que no siempre los modelos se adaptan a nuevas actualizaciones o cambios en los requerimientos [26].

6. Propuesta de arquitecturas agiles

La arquitectura ágil es un concepto relativamente nuevo que surge como reacción a las arquitecturas de software rígidas es decir arquitecturas que una vez diseñadas no sean flexibles a cambios ni actualizaciones, su objetivo es que el arquitecto de software y los demás interesados: cliente, desarrolladores etc. Esten presentes al momento del desarrollo de la arquitectura, para conocer diferentes opiniones y puntos de vista proporcionando una retroalimentación rápida lo que genera agilidad en el proceso de diseño además hace que todos estos interesados tengan una idea bien clara de cómo va a ser el funcionamiento del proyecto [29].

Se propone que la arquitectura no se elabore completamente antes de empezar el desarrollo del proyecto, sino que se haga continuamente mediante iteraciones, conforme avance el proyecto, esto para afrontar el posible cambio de negocio, cambios en los requerimientos y nuevas tecnologías, además se plantea que los componentes se encuentren en un nivel de desacoplamiento alto [28] para evitar que los cambios generen un gran impacto negativo.

7. Caso de estudio

El caso de estudio se desarrolló en la práctica donde la división de sistemas le asigna al estudiante el desarrollo de una plataforma para la gestión de torneos futbol entre las diferentes áreas de la empresa, se desea que el sistema posea las siguientes características: registro de usuarios, registro de equipos, programación de partidos, ver resultados de partidos y ver tabla de clasificación del torneo, además debe tener una interfaz agradable a la vista e intuitiva, además se debe redactar un manual de usuario para facilitar el uso de la plataforma. La practica del estudiante tiene una duración de 6 meses por lo que la plataforma debe estar lista en menos de este tiempo, el cliente que en este caso es el líder del área de deportes plantea que es una persona muy ocupada por lo tanto asignó un representante que siempre estará a disposición durante el proceso de desarrollo.

En la etapa de preparación se realizó una reunión inicial con el cliente y su representante. Ese día se conformó el equipo de trabajo y se fijaron requerimientos importantes sobre la plataforma. Se determinó que el equipo iba a tener tres miembros trabajando en conjunto y se realizó la siguiente distribución de roles:

- Experto de negocio: El representante del cliente.
- Desarrollador principal: El jefe del área de sistemas.
- Diseñador: El estudiante practicante.

Después de concluir la reunión se realizó se identificaron las prioridades y el experto de negocio propuso dos tipos de usuario dentro de la plataforma: usuario administrador y usuario común, cada uno con los siguientes objetivos:

Administrador: Registrar equipos, programar partidos, publicar resultado de partidos, ver resultados de partidos y ver tabla de clasificación.

Común: Registrarse, ver resultados de partidos y ver tabla de clasificación.

Basándose en lo anterior el usuario experto y el programador líder crearon la lista de actores – objetivos y a partir de esta confeccionaron un modelo de las tareas que el sistema debía realizar para satisfacer las necesidades de estos usuarios e identificaron y describieron los casos de uso del sistema que fueron añadidos al documento de requerimientos junto a la lista de actores – objetivos. En el documento de requerimientos se listaron los requisitos que el cliente deseaba que cumpliera el sistema.

Después de esta reunión, se realizó otra pequeña reunión en donde el desarrollador principal y el diseñador discutieron sobre que metodología se iba a implementar para el desarrollo del proyecto, se acordó que crystal clear gracias a sus propiedades y fácil implementación era la indicada para el proyecto, se decidió 3 entregas funcionales de 6 semanas y una planeación de 3 semanas en la que se diseñaría la arquitectura de software y asignarían las funcionalidades que tendría cada entrega, al finalizar cada entrega se tendrá una reunión que se llamara reunión de reflexión en la cual estaran todo el equipo de trabajo para conocer lo que se hizo en la entrega y lo que se hará en la siguiente para

así conocer la opinión del usuario experto y posiblemente obtener retroalimentación, el proyecto se desarrolló de la siguiente manera:

Planeación: El objetivo de esta era realizar la arquitectura de software e identificar que funcionalidades quedarían en cada entrega posterior, el desarrollador principal y el diseñador decidieron implementar la arquitectura dirigida por modelos gracias a que se podrían reutilizar modelos de otros proyectos realizados, además le brindaría al usuario experto un mejor entendimiento del software, en una visita del experto de negocio se realizo en conjunto el modelo CIM y se acordó que para cada entrega se debe realizar el modelo PIM y PSE con las funcionalidades correspondientes, además se decidió que el desarrollo se realizara en php con el framework symfony.

Entrega 1: Para esta entrega se definió que se crearán los dos tipos de usuarios y se harán los módulos de registrar usuarios y registrar equipos, se reutilizo el modelo PIM de registrar usuario de otro desarrollo creado anteriormente, se crearon el resto de modelos y adicionalmente se empezó a realizar el manual de usuario con estas funcionalidades, se empezó a codificar el proyecto, se hicieron las pruebas pertinentes, al final de esta entrega se hizo la reunión de reflexión y el usuario experto quedo satisfecho con el desarrollo, pero hizo una observación en el manual de usuario, dijo que se debe usar un lenguaje menos técnico pues muchas personas que usarían esta plataforma no entendían de computación.

Entrega 2: Para esta entrega se definió que se crearon los módulos de programar partidos y publicar resultados de los partidos jugados, se realizaron los módulos PMI y PSE correspondientes, así como su codificación, se hicieron las pruebas pertinentes, se corrigió el manual de usuario y se agregaron estas nuevas funcionalidades al manual de usuario, al finalizar en la reunión de reflexión el usuario experto ordeno que para la próxima entrega se agregara un módulo donde se pudieran ver las estadísticas de cada equipo tales como acumulación de tarjetas por cada jugador, goles por cada jugador entre otras.

Entrega 3: Para esta entrega se terminó el proyecto creando los módulos de ver tabla de clasificación y el nuevo módulo de ver estadísticas, se realizaron los modelos PMI y PSE correspondientes, así como su codificación, se hicieron las pruebas pertinentes, se pulió la interfaz para que fuera mas agradable y se termino el manual de usuario, se realizó la última reunión de reflexión donde se discutió el proceso seguido y las técnicas y estrategias aplicadas. Se guardaron las convenciones finales acordadas como referencia para futuros proyectos de software, el usuario experto quedo satisfecho con el trabajo realizado.

Al terminar se realizo una entrega con los miembros del equipo de trabajo y el cliente final en donde este reviso el manual de usuario e hizo una prueba exhaustiva del software y se le entrego totalmente funcional listo para ser usado, y así concluyó el caso de estudio.

Durante el desarrollo del caso de estudio se puede apreciar que en el proceso se realizan entregas funcionales a corto plazo, reuniones para reflexionar sobre el desarrollo, buena comunicación entre el equipo de trabajo y el cliente, disponibilidad por parte del usuario experto y facilidad de pruebas, siendo estas las propiedades de crystal, además los requerimientos y cambios no afectaron la arquitectura gracias a que se desarrolló continuamente, siendo una metodología recomendable, sencilla y fácil de implementar que es adaptable a cambios y facilita el diseño gradual de la arquitectura de software.

8. Conclusiones

Se ha podido observar en las diversas metodologías estudiadas que estas son un conjunto de guías, recomendaciones y pasos para administrar y llevar a cabo un proyecto de software y que estas juegan un papel muy importante para la culminación exitosa de un software y es que gracias a una buena aplicación de cualquier metodología ágil se obtiene un producto de alta calidad con los resultados esperados y por ende un cliente satisfecho, también se puede ver que en la arquitectura de software están involucradas estas metodologías ya que se presentan roles con responsabilidades similares a las de un arquitecto de software y también cuentan con fases para el desarrollo de la arquitectura.

La arquitectura de software es un proceso muy importante para el desarrollo de un proyecto de software ya que es como el esqueleto del software, un diseño bien estructurado ayuda a evitar problemas o situaciones que no fueron previstas, sirve como guía para los desarrolladores, mejora el entendimiento del software para las partes interesadas y ayuda a verificar que se cumplan todos los requerimientos y atributos de calidad.

De acuerdo con las técnicas investigadas, metodologías agiles y arquitectura de software, es muy difícil compararlas ya que una es parte fundamental de la otra para el desarrollo de un proyecto de software, pero se puede decir que para el éxito de un proyecto de estos, las metodologías agiles son más importantes ya que una buena implementación de estas ayuda a crear una arquitectura de software bien estructurada con todos los requisitos

funcionales y no funcionales, y por consecuente un producto final de muy alta calidad. gracias a su facilidad de implementación, la comunicación entre todos los miembros del equipo y la participación por parte del cliente que brinda una provechosa retroalimentación.

Lista de referencias

[1]http://www.funlam.edu.co/uploads/facultadpsicologia/76_Tipos_de_articulos_aceptados_para_publicacion.pdf

[2] Cervantes H. y Valencia E. (2011). *Integrando la Arquitectura al Ciclo de Desarrollo de Software.* México: Software Gurú.

[3] Guerra C. (2007). *Obtención de requerimientos. Técnicas y estrategia.* México: Revista Software Gurú.

[4] Lizcano L. (2002). *UML: Un lenguaje de modelo de objetos.* Cúcuta: Universidad Francisco de Paula Santander.

[5] Navarro A, Fernández J. y Morales J. (2013). *Revisión de metodologías agiles para el desarrollo de software.* Cali: Universidad Icesi.

[6] Canós J., Letelier P. y Penadés M. (2003). *Metodologías agiles en el desarrollo de software.* Valencia: Universidad politécnica de Valencia. Recuperado de: http://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf

[7] (2001) *Manifiesto por el desarrollo ágil.* Snowbird, Utah. Recuperado de: http://agilemanifesto.org/iso/es/manifesto.html

[8] Rivadeneria S. (2012). *Metodologías agiles enfocadas al modelado de requerimientos,* Provincia de Santa Cruz: Universidad Nacional de la Patagonia Austral.

[9] Amaya Y. (2013). *Metodologías agiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual,* Tunja: Universidad Pedagógica y Tecnológica de Colombia.

[10] http://www.scrumguides.org/scrum-guide.html#events

[11] Ladino M. (2007). *Mecanismo de consulta en línea sobre programación extrema (XP), scrum y crystal, metodologias agiles para el desarrollo de software, dirigido a estudiantes y progrsionales,* Pereira: Universidad Tecnológica de Pereira. Recuperado de: http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/791/0053L155mc.pdf?sequence=1&isAllowed=y

[12] Flórez L. (2014). *Formulacion de criterios para la selección de metodologias de desarrollo de software,* Pereira: Universidad Tecnológica de Pereira. Recuperado de: http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/5120/00512F634.pdf?sequence=1&isAllowed=y

[13] Botero N. (2015). *Monografía análisis e identificación de metodologías agiles y procedimientos específicos para la construcción de software.* Pereira: Universidad Tecnológica de Pereira. Recuperado de: http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/6013/00512B748.pdf?sequence=1&isAllowed=y

[14] https://sites.google.com/site/utmfci/home/ventajas-y-desventajas

[15] Clements P. (1996). *A survey of architecture description languages,* Alemania: Carnegie Mellon University. Recuperado de: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.625.6415&rep=rep1&type=pdf

[16] Cervantes H. (2010). *Arquitectura de software.* Mexico: Revista SG buzz. Recuperado de: https://sg.com.mx/revista/27/arquitectura-software#.WhWlfzdrzDc

[17] Valencia A., González M. (2011). *Documentación y análisis critico de algunas arquitecturas de software en aplicaciones empresariales.* Pereira: Universidad Tecnológica de Pereira.

[18] http://metodologiafdd.blogspot.com.co/

[19] Ferrer J. (2016). *Introducción arquitectura hexagonal -DDD.* Recuperado de: https://codely.tv/screencasts/arquitectura-hexagonal-ddd/

[20] https://www.britannica.com/technology/client-server-architecture

[21] Martin R. (2012). *The clean architecture.* Recuperado de: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

[22] Cockburn, A. (2001). *Agile software development.* Addison Wesley: Reading.

[23] Mariño C. & Puello A. (2014). *Metodologías crystal.* Santa Marta: Universidad del Magdalena. Recuperado de: https://es.scribd.com/doc/211905120/Metodologias-Crystal

[24] http://alistair.cockburn.us/Crystal+methodologies

[25] Venegas L. (2014). *Arquitectura manejada por modelos.* Ecuador: Revista San Gregorio. Recuperado de: https://dialnet.unirioja.es/servlet/articulo?codigo=5225646

[26] Den Hann J. (2009). *8 reasons why Model-Driven Development is dangerous* Recuperado de: http://www.theenterprisearchitect.eu/blog/2009/06/25/8-reasons-why-model-driven-development-is-dangerous/

[27] Cervantes H. (2011). *El rol del arquitecto de software*. México: Revista Software Gurú. Recuperado de: https://sg.com.mx/revista/33/el-rol-del-arquitecto-software#.WhwezDdrzDc

[28] https://proyectario.wordpress.com/2011/09/21/algo-sobre-arquitecturas-agiles/

[29] https://elfrasco.github.io/2015/06/08/Arquitectura-Agil.html

[30]